Overview

- Preliminaries.
  - Admin.
  - Questions on HW7.
  - Questions on Exam.
- Our sorting package.
- Testing sorts.
- Insertion sort.
- Selection sort.
- Lab.

Admin

- AA wants to know if anyone takes notes: MH
- Today we will do a few group exercises and then a few lab exercises.
- Upcoming extra credit opportunities:
  - Tonight’s Harry Hopkins talk, tonight at 7pm
  - Study in Budapest Lunch, Wednesday
  - Learning from Alumni, Thursday: Jordan Shkolnick ’11 (Microsoft)
  - CS Table, Friday: Ambient Belonging
  - One Grinnell Prize Event next week

Questions on HW7

Where do I find Node?

In DoublyLinkedList.java, because it’s only needed by that class.

What does search do?

Moves forward in the list until it finds a value for which the predicate holds. If it doesn’t find such a value, returns false and doesn’t move.

_Can I rewrite the Cursor interface so that it’s Cursor<T>?

Yes.

Can we work in groups of size 3?
Yes.

Questions on Exam

How should we submit?

Electronic version as attached tarball/zip

Our sorting package

● Two versions of sort, one in-place, one out-of-place
● It’s easy to turn an in-place algorithm into an out-of-place sorting algorithm
  ○ Clone the array
  ○ Sort the new array in place
  ○ Return it
● It’s easy to turn an out-of-place sorting algorithm into somethiung that simulates an in-place sorting algorithm (although it uses extra space)
  ○ Get the sorted version
  ○ Copy the values back
● You can see these strategies in practice in SorterBridge.java
● If you extend SorterBridge, and implement one of the two sorts, the other gets implemented "automagically"

Testing sorts

● Good testing involves automated generation of lots of cases
● And close attention to postconditions
● Randomized testing:
  ○ Generate a lot of random arrays
  ○ Sort them
  ○ Check postconditions
    ● It’s a permutation of the original - EXPENSIVE, PITN
    ● They’re in the correct order - EASY
● Can we avoid the "is it a permutation" check?
  ○ Use sequential integers
  ○ Start with a sorted "random" array. Then permute it. Then sort it.
  ○ Then compare.
● More systematic: Geneate every permutation of an array, sort it, then compare.
  ○ Think about this question for Wednesday
  ○ Goal: Do it "in place" - make a permutation, clone, sort, compare, go on to the next permutation
**Insertion sort**

- Divide array into sorted (nothing) and unsorted (everything)
- Repeatedly insert the thing at position $i$ into the sorted stuff at positions $[0..i)$
- Analysis: How long does this take:
  - $O(N)$ - Do something for each element. But each of those is not constant.
  - $O(N!)$ - Each insertion is $O(N)$. $O(N)$ of those. So $O(N^2)$
  - $1 + 2 + 3 + 4 + ...$ is also $O(N^2)$
- Sam’s old bad analysis:
  - At each step, we do binary search to find the right place
  - And it only takes one step to insert once you know the right place
  - Whoops! Insert is $O(N)$, even if you know the place
  - So $N*(\log N + 1)$ steps

**Selection sort**

- Divide array into sorted (nothing) and unsorted (everything)
- Repeatedly swap the smallest remaining element into the end of the sorted section
- Running time
  - $O(N)$ find smallest and swaps
  - Each find smallest is $O(N)$
  - So $O(N^2)$
- But only $O(N)$ swaps. Since writing memory is usually slow, cutting from $O(N^2)$ to $O(N)$ is good.

**Generate all Permutations**

- Goal *ALL* permutations
- Model: Some sort of loop or recursion that repeatedly
  - Makes a new permutation
  - Clones it
  - Sorts the new permutations
  - Does something (for testing, compare to original; for expt, print)
- You effectively have to make a loop for every position. How can we do that?
- If we could write the nest loop

```java
for (int i = 0; i < vals.length; i++) {
    // put the ith value in position (vals.length-1)
    // nested loops for positions [1 .. vals.length-2]
} // for
```

- So use recursion
Lab

- Clone https://github.com/Grinnell-CSC207/sorting
- Read code
- Finish implementing selection sort

Copyright © 2013 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.